Journal of Computer Science 7 (10): 1596-1604, 2011
ISSN 1549-3636

On the Fly Negotiation for Urgent Service Level Agreement on Intercloud Environment

Nawfal A. Mehdi, Ali Mamat,
Hamidah Ibrahim and Shamala K. Subramaniam
Department of Computer Science,
Faculty of Computer Science and Information Technology,
University Putra Malaysia, 43400 Serdang, Malaysia

Abstract: Problem statement: Impatient jobs are the jobs that have strict constraints in starting and completion time and need a fast response and attention in mapping to resources. The negotiation process in cloud computing needs a customer-provider agreement that leads to loss of a significant amount of time for impatient jobs. The huge increasing in cloud service providers makes selecting the best provider an exhaustive task. **Approach:** We considered the problem of Service Level Agreement (SLA) negotiation and commitment process for immediate mode scheduling under intercloud paradigm. This study explored an alternative model dedicated for impatient jobs under intercloud paradigm. We developed a model of negotiation where the cloud-broker had the ability to nominate the best cloud provider, commit the agreement and submit the jobs for execution. Cloudsim simulators with synthetic datasets had been used to evaluate the proposed system. **Results:** The results showed an improvement in SLA waiting time and the number of jobs failure. **Conclusion:** This study proved the importance of rapid mapping for impatient jobs in increasing system throughput.

Key words: Cloud computing, Service Level Agreement (SLA), impatient job, virtual machine, negotiation process, negotiation algorithm, Quality of Service (QoS), Physical Machines (PM)

INTRODUCTION

Cloud computing (Michael *et al*., 2009) is the new paradigm of computing where easily offers computing resources as services. These computing services are generally charged using a pay-as-you-go pricing method and hence, it becomes attractive to most customers. Cloud computing can be defined as Internet-based "cloud" services and use of computer technology (computing) that offers flexible and dynamic IT infrastructure, a Quality of Service (QoS) (Yin *et al*., 2010) guaranteed environment and reconfigurable services (Wang *et al*., 2008). This new paradigm is being driven by many well known cloud providers like Roebuck (2011); Miller (2011) Microsoft (Jennings, 2010; Lohr, 2007) HP Cloud Services and IBM Cloud Services

Multiple clouds can interoperate with each other to form what is called Intercloud (Bernstein *et al*., 2009). It can be defined as "cloud of cloud" (Metz, 2009; Briscoe and Marinos, 2009; Johnston, 2009) and it is a metaphor for Internet, which is Network of Networks. Intercloud was first coined by Kelly (2007) in his article. "Eventually we'll have the intercloud, the cloud of clouds", Kelly writes.

SLA (Pichot *et al*., 2009; Hovestadt, 2006) is a contract signed between a service provider and a customer that describes the service, responsibilities, terms, guarantees and service-level to be provided. SLA is an important element of the service oriented computing paradigm and defines a mutually agreed upon set of consumer expectations and provider obligations. Typically, SLAs encode QoS parameters such as resource availability, response time and completion deadlines. The role of the consumer is usually limited to specify their QoS parameters and perhaps revising those parameters if an SLA cannot be agreed (Netto *et al*., 2010).

According to Jennings *et al*. (2001), a negotiation can be defined as: "the process by which a group of agents come to a mutually acceptable agreement on some matter". SLA Automatic negotiation can be a complex and time-consuming process when two parties need to create an agreement on multiple criteria (Jennings *et al*., 2001; Shen *et al*., 2002).

Corresponding Author: Nawfal A. Mehdi, Department of Computer Science, Faculty of Computer Science and Information Technology, University Putra Malaysia, 43400 Serdang, Malaysia

Cloud computing providers offer their services after confirming SLA agreement between them and the customers. The negotiation process needs both parties to agree on the SLA and then sign the contract. This process involves many steps, which is sometimes time consuming for some customers with urgent needs. The normal SLA negotiation process requires sending an offer from the customer (i.e., buyer) to the provider (i.e., seller) and the response by a bid from the provider to the customer. This process can be repeated several times until an agreement is reached between both sides. In the last step, the provider creates the agreement template and sends it back to the customer for confirming. The customer then confirms the template. The last step is time-consuming and is only needed to confirm the offer again (i.e., sign the contract).

In addition to above, the huge growing in cloud providers and their services increase the difficulty for urgent customers to surf all their services and provide a decision to which the request should be sent. Most of the previous negotiation strategies include the steps of sending an offer from the customer (i.e., client or buyer) to the provider (i.e., seller) asking for a specific Service (s), the provider returns with their bid or acceptance of the customer's offer and the last step is the confirmation from the provider side.

On-the-Fly-Negotiation Algorithm (OFNA) is the proposed algorithm that maps the jobs to the cloud services with less negotiation steps to provide fast response for urgent jobs. The proposed model takes into consideration the urgent and normal jobs.

Related work: Since the 1980s, SLAs were established as tools for stating the QoS. They were mainly used in the telecommunication domain and used as study print-outs there was a tendency in the research community to try to adapt the SLA concepts on other domains (Green *et al*., 2007; Munoz *et al*., 2010).

Work presented Al-Ali *et al*. (2002), extended the service abstraction in the Open Grid Services Architecture for QoS properties. They focused on the application layer, whereby a given service may indicate the QoS properties it can offer, or where a service may search for other services based on particular QoS properties.

Ouelhadj *et al*. (2005) proposed a new infrastructure for efficient job scheduling on the Grid using multi-agent systems and a SLA negotiation protocol based on the Contract Net Protocol. In their protocol, the agents exchange SLA-announcements, SLA-bids and SLA-awards to negotiate the schedule of jobs on Grid compute resources. Their model needs multi-level negotiation between the agents. Furthermore, the client in their model has to select a bid from set of offers and commits it.

Munoz *et al*. (2010) and Parkin *et al*. (2008) described an abstract, domain-independent protocol for the renegotiation of an agreement, including SLA formed using the WS-Agreement standard. Their proposed protocol is based on the principles of contract law to make any new agreements using them, legally compliant. It allows for multi-round renegotiation in a network environment where messages may be lost, delayed, duplicated and re-ordered. In their mode, the user needs pre-knowledge about the current services and commits the last agreement.

Work presented Green *et al*. (2007) proposed novel augmentations to existing service negotiation protocols in the areas of scalability, flexibility, support for distinct services and negotiation with several service providers simultaneously. Their proposed autonomous negotiation protocol is based on a distributed multi-agent framework creating an open market for Grid services. Their model includes at each negotiation process a binding stage whereby a valid SLA instance is formally agreed to by the consumer and all the involved service providers. After binding, the consumer and provider(s) have to commit the agreed SLA.

Pichot *et al*. (2008) in their study, discussed basic functionality for resource orchestration in grids, namely mechanisms to dynamically negotiate and create service level agreements using WS-Agreement. They proposed multi-level negotiation process where the meta-scheduler should negotiate with the provider to find the best template. The SLA should be committed using two-phase commit protocol.

The proposed framework Hudert *et al*. (2009) augments this WS-Agreement to enable negotiations according to a variety of bilateral and multilateral negotiation protocols. The framework design is based on a thorough analysis of taxonomies for negotiations from the literature in order to allow for capturing a variety of different negotiation models within a single WS-Agreement compatible framework. In order to provide for the intended flexibility, the proposed protocol takes a two-stage approach: a meta-protocol is conducted among interested parties to initially agree on a common negotiation protocol before the real negotiation is carried out in the second step using the protocol established in the first step.

Work presented An *et al*. (2010) considered the problem of allocating networked resources in cloud computing platforms, where providers strategically price resources to maximize their utility. They explore an alternative approach where providers and consumers automatically negotiate resource leasing contracts. In

their model, both the provider and consumer are selfish. The consumer needs to know the ability of each cloud provider (i.e., pre-knowledge) and to commit the agreement before approval.

MATERIALS AND METHODS

In the proposed model, there are three tiers namely: the customer, the cloud broker and the cloud providers. Each cloud provider has set of datacenters which in turn consists of set of Physical Machines (PM) used to host Virtual Machines (VMs). The cloud broker acts as an advisor for the customers. The customer c sends his request R_c, which has the list of jobs J_c, to broker B. The customer does not know which cloud provider is suitable to execute his request or he does not have the time to look for the appropriate provider.

The main idea of this study is to let the cloud broker commit and submit the impatient jobs to the cloud provider. The customer should permit the broker agent to sign the agreement on behalf of the customer if the broker agent finds a suitable cloud provider that fits with the requirements of the customer's QoS.

We assume that customer c and the broker B are working on a utility function as described in (1).

$$U_c = \{\min(J_c), \min(\beta_c)\} \quad (1)$$

Where:
- U_c = Denotes the utility function of customer c
- J_c = Denotes the number of deadlines (start and complete) that do not meet their requirements
- β_c = Denotes the total budget for all the jobs in request R_c

The time needed to execute each job j_c in the request R_c is computed using (2):

$$TET_{jc}^{up} = stagein_{jc}^{filein} + E_{jc}^{up} + stagein_{jc}^{fileout} \quad (2)$$

where, TET_{jc}^{up} denotes the time needed to execute the job j_c sent by customer c and nominated to execute on VM v_p, which is offered by provider p. $stagein_{jc}^{filein}$ is the time needed to fetch all the necessary input files while $stagein_{jc}^{fileout}$ is the time for sending out all the output files to their destinations. E_{jc}^{up} gives the time needed to execute the job on the nominated virtual machine.

The estimated job start time is calculated based on the current time and other factors as shown in (3):

$$EST_{jc}^{up} = CT + N.(\delta_c + \rho) + (1 - A_c)\omega_c \quad (3)$$

Where:
- CT = The current time
- N = The number of counteroffers between the customer c and broker B such that $N \geq 1$

The time needed by customer c to create the request R_c and send it to the broker B is equal to δ_c. The time needed by broker B to map all the jobs in R_c to the available resources and return back the offer is equal to ρ. The estimated time for customer c to confirm the offer is ω_c and A_c is the decision variable such that:

$$A_c = \begin{cases} 1, \text{if c sets auto} - \text{confirmation} \\ 0, \text{otherwise} \end{cases} \quad (4)$$

As can be seen from (3), if the customer c sets the auto-confirm option, the broker does not consider the confirmation time and vice versa. The completion time for each job j_c is proposed to be computed using (5) as shown:

$$FT_{jc}^{vp} = EST_{jc}^{vp} + TET_{jc}^{vp} \quad (5)$$

Now let x_{jc}^{start} be the decision variable that indicates whether the start time meets with the job requirements or not.

$$x_c^{start} = \begin{cases} 1, \text{if } EST_{jc}^{vp} > sd_{jc} \\ 0, \text{otherwise} \end{cases} \quad (6)$$

and let $x_{jc}^{completion}$ be the decision variable that indicates whether the completion time meets with the job requirements or not.

$$x_{jc}^{completion} = \begin{cases} 1, \text{if } FT_{jc}^{v_p} > dl_{jc} \\ 0, \text{otherwise} \end{cases} \quad (7)$$

Now, from (6) and (7) we can find the utility variable J_c as shown below:

$$J_c = \sum_{j \in Jc} x_{jc}^{start} + \sum_{j \in Jc} x_{jc}^{completion} \quad (8)$$

Customer c can ask for full respect of his request R_c, which means $J_c = 0$ or he can specify a maximum value for it.

While most of the current cloud providers charge the customers per hour, we can compute the amount of total charge for customer c and his request R_c by:

$$\beta_c = \sum_{j \in J} \frac{TET_{j_c}^{vp}}{3600} * cost_v^p \quad (9)$$

where, β_c is the total amount of money that should be paid by costumer c to the cloud provider(s). It is calculated by multiplying the VM cost by the total execution time in hours as can be seen in (9).

Negotiation protocol: The architecture of our proposed negotiation framework is shown in Fig. 1. It is composed of three kinds of agents that are responsible for managing the negotiation process and creating the SLA agreements. These agents are:

Customer agent: This agent is located at the customer's side. Its responsibilities are: (a) collects the list of jobs and their requirements, (b) creates the request template and (c) sends the request template to the broker agent and waits for acknowledgment.

Broker agent: This agent is the mediator between the cloud customers and cloud providers. It can also be considered as the meta-scheduler of the cloud customers' requests. Its main responsibilities are: (a) receives the request template from the customers agent and decodes it, (b) sends the requests to the scheduler that can find the best mapping for this request, (c) creates the agreement template and signs it on behalf of the customer in case of auto-confirmation or sends it back to the customer's agent, d) submits the customer's request to the cloud providers and (e) receives the cloud updates from the provider agent, which is the current status of each cloud provider to be used by the scheduler.

Cloud provider agent: This agent is located at the side of each cloud provider and has many responsibilities, which are: (a) periodically sends the status of the cloud to the broker agent and (b) receives the list of jobs to be executed from the broker agent.

Agents actions: The main actions that are proposed to be taken by the agents are:

- Request[r]: The work flow starts from the customer c side by sending its request R_c, which has the list of job J_c to the broker agent B
- Submit[r]: If the scheduler finds a suitable mapping for the customer's request, the broker agent passes this request to the cloud providers to execute them and simultaneously send a confirmation message to the customer's agent
- Reject[r]: When the scheduler cannot find a mapping that meets the customer's requirements from the deadline and budget point of view, the broker agent sends a reject message back to the customer agent for refining
- Bid: When the broker agent B meets the deadline restrictions but with more budget requirement, a bid offer O is sent back to the customer agent c
- Decommit: Decommit is defined as a withdraw from active service. This action is done by the customer c to cancel the current offer O or the current request R_c

Action flow sequence: The process of SLA negotiation can be defined as an activity to determine certain details of an interaction. Figure 2 depicts the finite state machine for the process of negotiation between the cloud customer agent, the broker agent and the cloud provider agent.

Initially, the cloud customer c, who has an impatient job(s), initiates the request r by creating a template that describes all the required jobs and their properties. It includes all the job constraints including the deadlines and budgets.

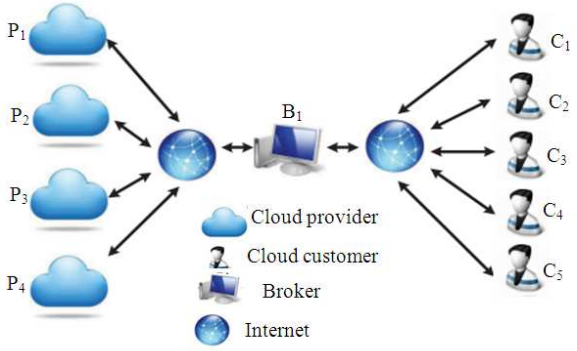

Fig. 1: The negotiation framework architecture

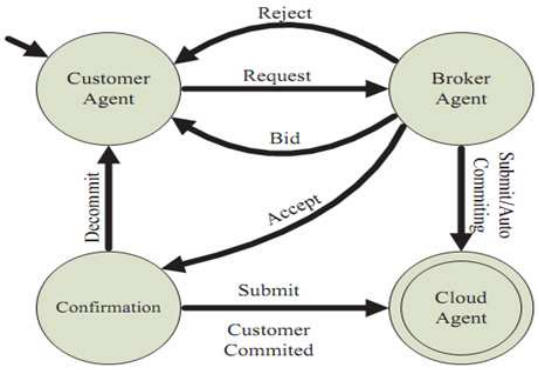

Fig. 2: Finite state machine for the proposed system

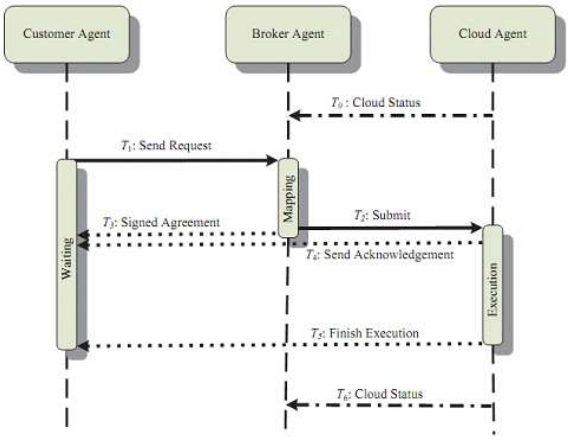

Fig. 3: Work flow sequence in the proposed model

```
1  foreach j ∈ J do
2      if j is critical job then
3          A_j ← auto;
4      else
5          A_j ← commit;
6      Add j and its requirements to R_c;
7  Send(R_c);
8  if broker agent b accepts request R_c then
9      if A_r ≡ auto then
10         Wait(P,Ack);
11     else
12         CheckOffer(O);
13 else
14     Cancel(r);
15 if provider P sends acknowledgment then
16     Wait(P,result);
17 if provider P sends results then
18     Finish;
```

Fig. 4: Algorithm of customer strategy

The broker agent B receives the request r from the customer agent c. The broker sends r to the meta-scheduler to find the best mapping that can respect all the job requirements. If the mapping is done (i.e., respecting all the QoS requirements) then it has two directions. The first direction is to submit the mapping list directly to the nominated cloud provider without waiting for customer c conurbation in the case of auto-conurbation, which is an option selected by the customer c. If the customer asks for the normal negotiation process (i.e., without auto-confirmation), the broker agent sends the agreement to the wait state, which is the state that awaits the commitment from customer c. In the waiting state, the customer has the ability to commit or reject the agreement. In case of commitment, the broker agent B sends the request to the cloud provider p to execute, otherwise the whole process returns to the initial state.

Events workflow of negotiation process: Figure 3 depicts the study flow sequence of the negotiation process. It shows the three agents (customer c, broker B and provider p). In the proposed model, the cloud provider agent p sends the cloud status periodically as initially occurs at time J_0. These values (i.e., templates) describe the current state of the cloud by the number of available virtual machine images, the specifications of each VM from SW, HW and the cost point of view and the dynamic information such as the availability. At time J_1 the customer agent c sends the request r to the broker agent who has the list of jobs and their QoS requirements. The next time event J_2 is for broker agent B to submit the request r to the nominated cloud provider. This action is done in case of auto-conurbation permission, which is taken form customer agent c. The broker agent at time J_3 sends an acknowledgement to the customer agent concerning the status of agreement.

This acknowledgment is sent after submitting the request to the cloud provider to save time. Time j_4 is the time to start execution of the customer's request r, thus, it is proposed for the cloud to acknowledge the customer about this action.

After the cloud provider finishes its execution, it should either send the result back to the customer or to the third party and acknowledge the customer at the same time.

Agents strategy: As the aforementioned model shows the three components of intercloud paradigm, the three agents namely customer strategy, broker strategy and provider strategy are illustrated below.

Customer strategy: The customer's algorithm is depicted in Fig. 4. Steps (1-6) are used by the customer c to create the request R_c to be sent to the broker. The if statement at step 2 is used to set the control variable A with or without value. We focus on this variable because it plays an important role in the proposed system. After creating the request R_c the customer c

sends the request to broker B. This is done at step (7). The statement at step (8) is used to check whether the broker accepts the requests or not. Two actions exist if the broker B accepts the request. If the broker accepts the request while the customer c sets the variable A to auto then it should wait for the acknowledgment from the provider, otherwise (i.e., A is not auto) the customer has to check the offer.

```
1  if customer c sends request R_c then
2      T_c, B_c ← CMCT(r);
3  if T_c ≡ 0 then
4      if B_c ≤ β_c then
5          if A_r ≡ auto then
6              Submit (R_c, P);
7          else
8              Wait (c, conformation);
9      else
10         Send(c, P, T_c, B_c );
11 else
12     Send (c, "reject");
13 if customer c confirm then
14     Submit (R_c , P);
15 if customer c dicommit then
16     Send (c, "reject");
17 if provider p sends cloud update then
18     Update (cloud p Status)
```

Fig. 5: Algorithm of broker strategy

```
1  foreach μ time interval do
2      Send (B, Status)
3  if Broker B submits requests R_c then
4      Execute (R_c)
5  if Provider p finishes execution request R_c then
6      Send (c, results)
```

Fig. 6: Algorithm of provider strategy

In case the broker rejects the offer for its impossibility, then the customer should cancel the submitted request R_c as can be seen in step (14). Step (15) and (16) check the case if the provider p sends an acknowledgment to the customer c and calls the wait function. If the result is received by the customer, then customer agent c finishes its work, as can be seen in steps (17) and (18).

Broker strategy: The broker algorithm is depicted in Fig. 5. The broker keeps listening to two things: (a) requests from the clients and (b) cloud status from the cloud providers. Steps (1) and (2) are responsible for listing the clients' requests and sending them to the scheduler. In this study we consider the Minimum Completion Time (MCT) algorithm as immediate mode scheduling algorithm.

It is adopted to be compatible with the cloud environment and renamed it to Cloud Minimum Completion Time (CMCT). In step (2), the scheduling algorithm returns the number of failed to meet deadlines, which is denoted by J_c and the estimated execution price, which is denoted by β_c. If all the jobs meet their deadlines (i.e., step (3)) then the request is either submitted directly to the nominated cloud provider if the customer sets A as auto (steps (5) and (6)) or waits for the customer to confirm the offer (see step (10). In case of failure in some deadlines then step (12) sends a reject message to the customer. Step (14) submits the request to the cloud provider if the customer confirms. Step (16) sends a reject message to the customer in case the latter decommit the offer. Cloud providers periodically send their cloud status. Step (18) updates the cloud status.

Provider strategy: This study proposes that each cloud provider has an agent to deal with the broker agent. Figure 6 shows the main steps of this agent. Let μ be the time interval to send the cloud status to the broker as can be seen in steps 1 and 2. Steps 3 and 4 are responsible for request execution if the broker B sends as request. Sending the results back to the customer c is at step 5 and 6.

RESULTS

In the absence of real traces from real cloud providers, we generated the input workload randomly. Casazza *et al*. (2006) present a workload methodology to characterize the performance of servers exploiting virtualization technologies to consolidate multiple physical servers. They combine the workload of the

web server, an email server and a database application to reflect the variation of application that can be run on cloud computing.

As aforementioned, we assume a certain number of users, jobs and cloud providers. The job workload is selected randomly (Ranganathan and Foster, 2002) with uniform distribution between 500-2000MB. Each job has a set of input and output files selected randomly between 1 and 6. Job length is a function to the total input size for 300D if we assume D is the total input size. For simplicity and without loss of generality we assume all jobs need a fixed time to offer or bid and each job needs 0..5 counteroffers. The job deadlines are functions that are based on their length and their data size. For the purpose of this work, the deadlines are very strict and hard, which means the jobs need quick attention.

Simulation is the process of imitation of the real system. Because of the difficulty in testing the proposed system in a real system, a simulation evaluation has been conducted on synthetic datasets. CloudSim (Calheiros *et al*., 2009) is a discrete event simulator that is used to simulate cloud environments. Cloudsim has the ability to create data centres, virtual machines and physical machines and configure system brokers, system storage.

Table 1 specifies the simulation parameters used for our study. Two performance metrics have been used to evaluate the proposed model namely: number of jobs failure and average waiting time. To evaluate the performance of the proposed model, twenty experiments were done using the cloudsim simulator. We created ten datasets with a different number of jobs and different loads to evaluate the performance.

As aforementioned, the deadlines are functions of the job's length and their data. The simulation process is done to evaluate the impact of on-the-fly algorithm on impatient job mapping under intercloud paradigm.

Figure 7 depicts the number of job failures in the proposed model and the CMCT. It indicates the number of jobs for which the scheduler cannot meet their deadlines. The value of this metric is the inverse to the value of throughput that indicates the number of finished jobs. The figure shows some failure even with the proposed system, which is because of the hard deadlines created within the synthetic datasets. The improvement in the system throughput using the proposed model is quite clear. Also, it is possible to notice that as the number of submitted jobs increase, the number of failed jobs is also increase. This is because of start deadline constraint that failed to meet the jobs requirements which is because of more waiting time through confirmation step.

Figure 8 depicts the average waiting time for all the submitted jobs. These metric measures the time needed to finish the negotiation process. It is the difference between the job arrival-time to the starting time of the scheduler to map this job as shown in Eq. 12. In this Fig. 8, we compute the waiting time for all the jobs even the failed jobs.

$$AWT_{jc} = \frac{1}{|Jc|}\sum_{j\in Jc}(st_j - at_j) \quad (12)$$

Where:

AWT_{Jc} = The average waiting time for job list Jc

st_j = The time for the job j to reach the scheduler (i.e., finish the negotiation process)

Figure 8 shows the improvement that happened in the waiting time or the negotiation time. The effect of confirmation on the total negotiation time is quite clear. The need for confirmation can improve the system based on the user's response time which implemented randomly in this study.

Table 1: Cloudsim configurations

Item	Value
Number of datacenters	14
Number of VM	100
Number of CPU/VM	1
CPU Speed/VM	1, 2, 2.5 and 3 GHz
Number of tasks	5,10,15,20,30,50,100,200,300,400

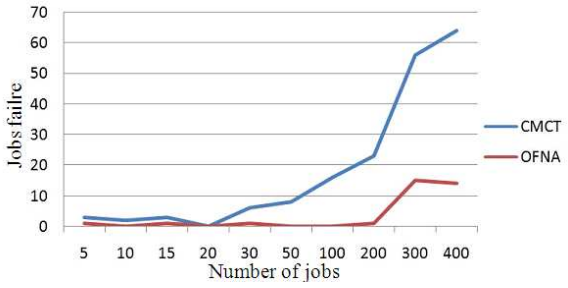

Fig. 7: Job failure among different sets of jobs

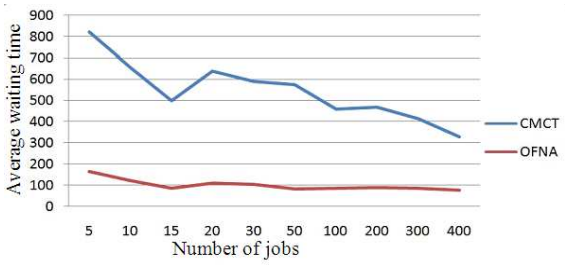

Fig. 8: Average waiting time among different set of jobs

DISCUSSION

The SLA negotiation is the bargaining process between the cloud service provider and the cloud service client (i.e., consumer). The urgent client needs fast attention to speed up the process of its execution. The negotiation process steps can be reduced to save the client's time if both agents agree. This can be done if the urgent client trusts the cloud provider and agrees to let the provider to select services and forward it the task on behalf of the client. This way can reduce an amount of time that is precious and can increase the total system throughput.

CONCLUSION

This study tackles the negotiation process and tries to minimize the number of steps needed for agreement to minimize the waiting time for impatient jobs. The job waiting time depends on the number of counteroffers and the confirmation steps if we assume there are enough resources. System throughput can be increased for impatient jobs if we can offer quick attention and less response time. We can conclude from these experiments that the commitment step is an expensive process to urgent jobs.

REFERENCES

Al-Ali, R., O. Rana, D. Walker, S. Jha and S. Sohail, 2002. G-QoSM: Grid service discovery using QoS properties. Comput. Inf., 21: 363-382.

Amazon Elastic Compute Cloud, 2011. http://aws.amazon.com/ec2.

An, B., V. Lesser, D. Irwin and M. Zink, 2010. Automated negotiation with decommitment for dynamic resource allocation in cloud computing. Proceedings of the 9th International Conference on Autonomous Agents and Multiagent System, (AAMAS '10), International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, pp: 981-988.

Bernstein, D., E. Ludvigson, K. Sankar, S. Diamond and M. Morrow, 2009. Blueprint for the intercloud-protocols and formats for cloud computing interoperability. Proceedings of the 4th International Conference on Internet and Web Applications and Services, IEEE Xplore Press, Venice/Mestre, pp: 328-336. DOI: 10.1109/ICIW.2009.55

Briscoe, G. and A. Marinos, 2009. Digital ecosystems in the clouds: Towards community cloud computing. Proceeding of the 3rd IEEE International Conference on Digital Ecosystems and Technologies, June 1-3, IEEE Xplore Press, Istanbul, pp: 103-108. 10.1109/DEST.2009.5276725

Calheiros, R., R. Ranjan, C. De Rose and R. Buyya, 2009. CloudSim: A novel framework for modeling and simulation of cloud computing infrastructures and services. Cornell University Library.

Casazza, J., M. Greeneld and K. Shi, 2009. Redefining server performance characterization for virtualization benchmarking. Intel Technol. J., 10: 243-251. DOI: 10.1535/itj.1003.07

Green, L., V. Mirchandani, I. Cergol and D. Verchere, 2007. Design of a dynamic SLA negotiation protocol for grids. Proc. Int. Conf. Networks Grid Appl.

Hovestadt, M., 2006. Operation of an SLA-aware grid fabric. J. Comput. Sci., 2: 550-557. DOI: 10.3844/jcssp.2006.550.557

Hudert, S., H. Ludwig and G. Wirtz, 2009. Negotiating SLAs An approach for a generic negotiation framework for WS-Agreement. J. Grid Comput., 7: 225-246. DOI: 10.1007/s10723-009-9118-3

Jennings, N., P. Faratin, A. Lomuscio, S. Parsons and M. Wooldridge *et al*., 2001. Automated negotiation: Prospects, methods and challenges. Group Decis. Negot., 10: 199-215. DOI: 10.1023/A:1008746126376

Jennings, R., 2010. Cloud Computing with the Windows Azure Platform. 1st Edn., John Wiley and Sons, New York, ISBN: 1118058755, pp: 360.

Johnston, S., 2009. The Inter cloud is a global cloud of clouds. Sam Johnston.

Kelly, K., 2007. A cloudbook for the cloud. The Technium.

Lohr, S., 2007. Google and IBM join in cloud computing research. New York Times.

Mehdi, N.A., A. Mamat, H. Ibrahim and S.K. Subramaniam, 2011. Impatient task mapping in elastic cloud using genetic algorithm. J. Comput. Sci., 7: 877-883. DOI: 10.3844/jcssp.2011.877.883

Metz, C., 2009. The Meta Cloud-flying datacenters enter fourth dimension. The register.

Michael, A., F. Armando, G. Rean, D. Anthony and K. Randy *et al*., 2009. Above the clouds: A berkeley view of cloud computing. University of California.

Miller, M., 2011. Using Google Apps. 1st Edn., Pearson Technology Group, Indianapolis, ISBN: 0789743973, pp: 298.

Munoz, H., I. Kotsiopoulos, A. Micsik, B. Koller and J. Mora, 2010. Flexible SLA Negotiation Using Semantic Annotations, Service-Oriented Comput., 6275: 165-175 DOI: 10.1007/978-3-642-16132-2_16

Netto, M., K. Bubendorfer and R. Buyya, 2010. SLA-based advance reservations with flexible and adaptive time QoS parameters. Service Oriented Comput., 4749: 119-131. DOI: 10.1007/978-3-540-74974-5_10

Ouelhadj, D., J. Garibaldi, J. MacLaren, R. Sakellariou and K. Krishnakumar *et al*., 2005. A multi-agent infrastructure and a service level agreement negotiation protocol for robust scheduling in grid computing. Adv. Grid Comput., 3470: 651-660. DOI: 10.1007/11508380_66

Parkin, M., P. Hasselmeyer, B. Koller and P. Wieder, 2008. An SLA re-negotiation protocol. Sit Seerx Beta.

Pichot, A., O. W. Aldrich, W. Ziegler and P. Wieder, 2009. Towards Dynamic service level agreement negotiation: An approach based on WS-agreement. Web Inf. Syst. Technol., 18: 107-119. DOI: 10.1007/978-3-642-01344-7_9

Pichot, P. Wieder, O. W Aldrich and W. Ziegler, 2008. Dynamic SLA-negotiation based on WS-agreement. Proceedings of the 4th International Conference on Web Information Systems and Technologies, May 4-7, Funchal, Madeira-Portugal, pp: 38-45.

Ranganathan, K. and I. Foster, 2002. Decoupling computation and data scheduling in distributed data-intensive applications. Proceeding of the 11th International Symposium on High Performance Distributed Computing, July 23-26, IEEE Xplore Press, USA, pp: 352-358.

Roebuck, K., 2011. Amazon Elastic Compute Cloud (EC2): High-impact Strategies - What You Need to Know: Definitions, Adoptions, Impact, Benefits, Maturity, Vendors. 1st Edn., Emereo Pty Limited, USA, ISBN**:** 1743046979, pp: 326.

Shen, W., Y. Li, H. Ghenniwa and C. Wang, 2002. Adaptive negotiation for agent-based grid computing. University of Western Ontario

Wang, L., J. Tao, M. Kunze, A. Castellanos and D. Kramer *et al*., 2008. Scientific cloud computing: Early definition and experience. Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications, Sept. 25-27, IEEE Xplore Press, Dalian, pp: 825-830.

Yin, K., B. Zhou, S. Zhang, H. Jiang and J. Cristoforo, 2010. Optimizing services composition in multi-network environment. Inf. Technol. J., 9: 399-411.